Becoming an Intermediate Python Developer

Organized by University of Manitoba IEEE

Your UMIEEE Workshop Presenter

Mahyar Mirrashed

Topics to be Covered (Agenda)

- Python f-strings for effective string formatting
- Python lambdas (anonymous functions)
- Python list, dictionary, and set comprehensions
- Python function decorators and inner functions

Approximated length: 50-60 minutes. Extra 10 minutes at end for questions.

Getting the Most From Online Workshops

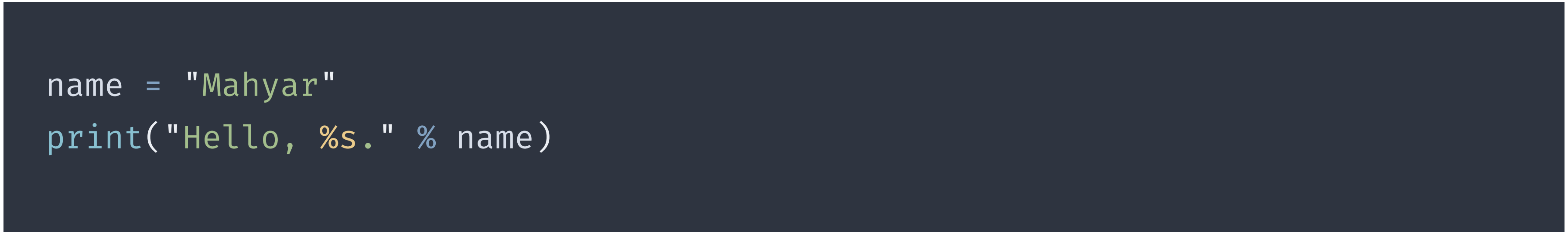
- Take notes!
- Ask questions in Google Meet chat!
- Follow along if possible!

Take advantage of this UMIEEE workshop to the greatest extent possible!

f-Strings for Effective String Formatting

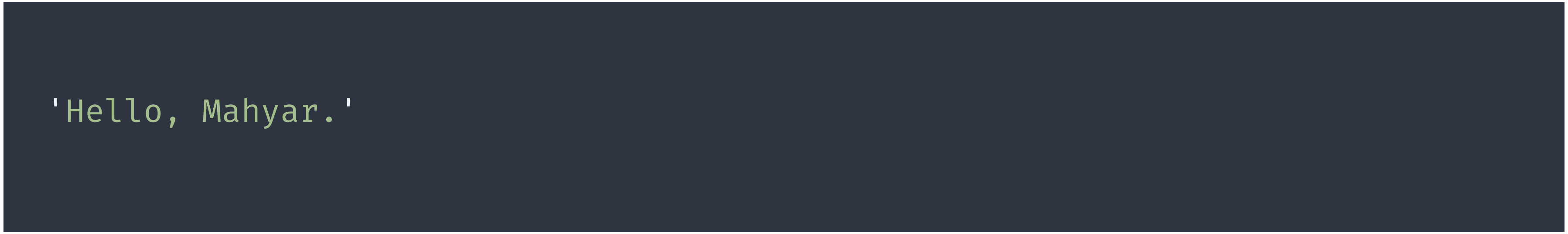
“Old-school” String Formatting: %-formatting

INPUT:



```
name = "Mahyar"
print("Hello, %s." % name)
```

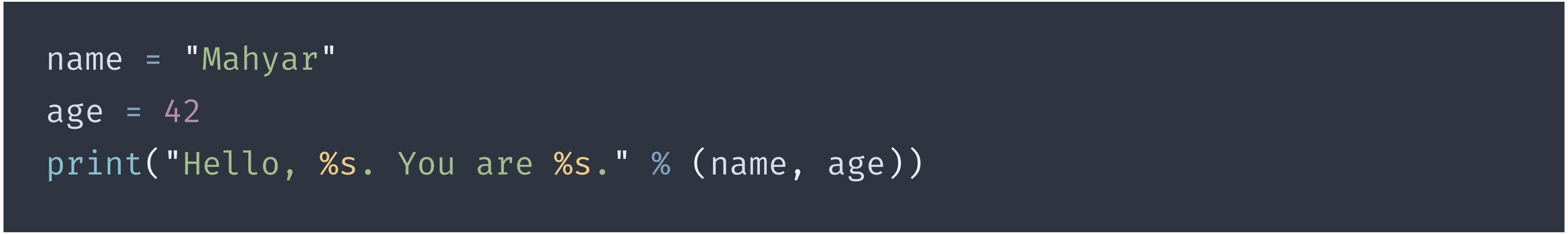
OUTPUT:



```
'Hello, Mahyar.'
```

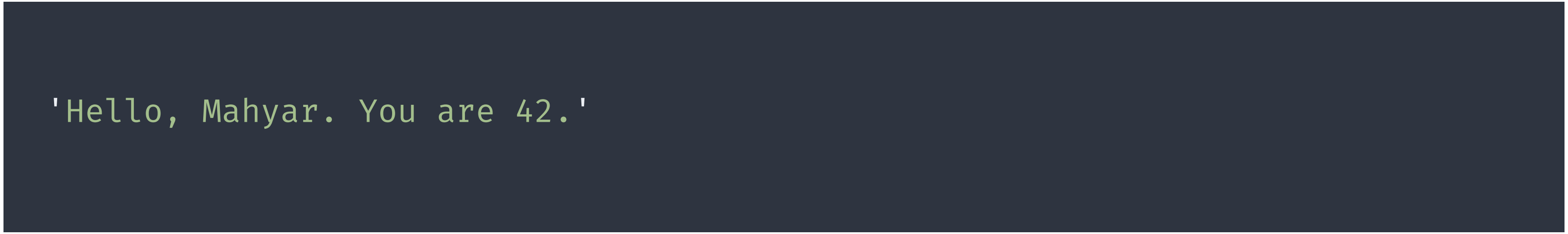
“Old-school” String Formatting: %-formatting

INPUT:



```python
name = "Mahyar"
age = 42
print("Hello, %s. You are %s." % (name, age))
```

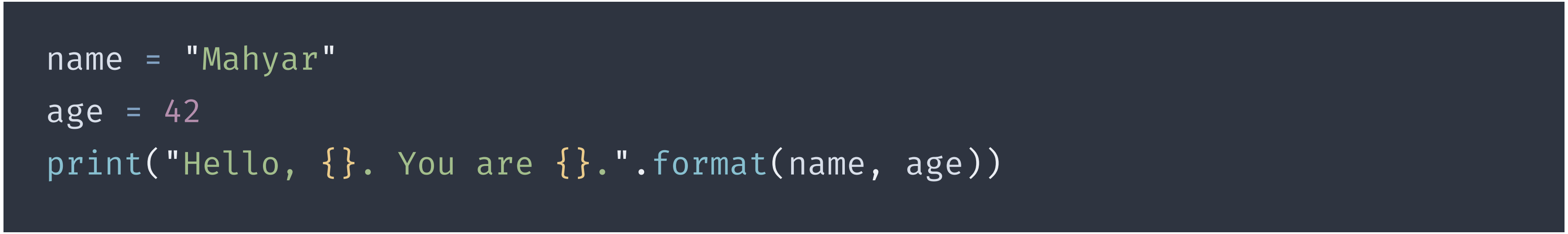
OUTPUT:



```
'Hello, Mahyar. You are 42.'
```

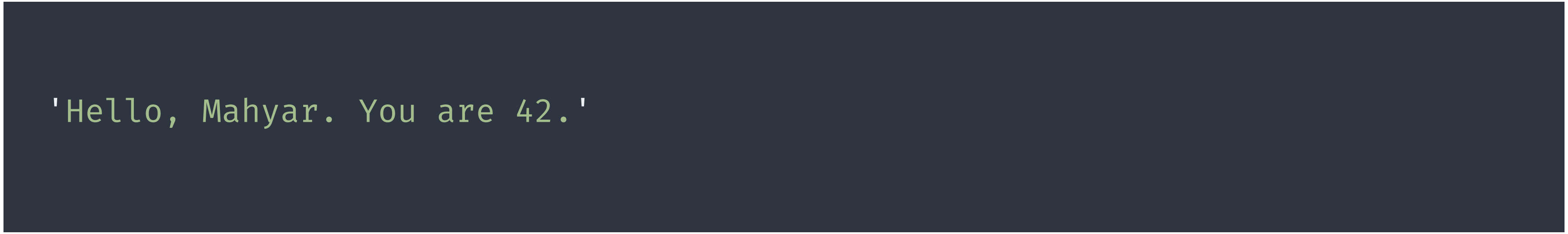
“Old-school” String Formatting: `str.format()`

INPUT:



```
name = "Mahyar"
age = 42
print("Hello, {}. You are {}.".format(name, age))
```

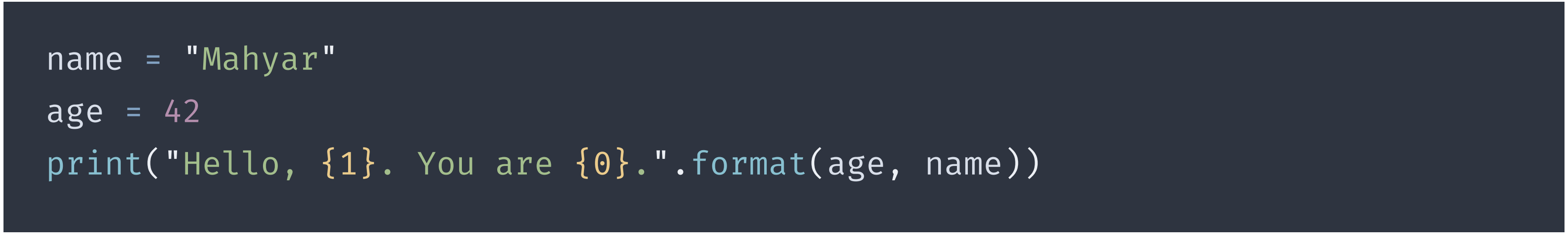
OUTPUT:



```
'Hello, Mahyar. You are 42.'
```

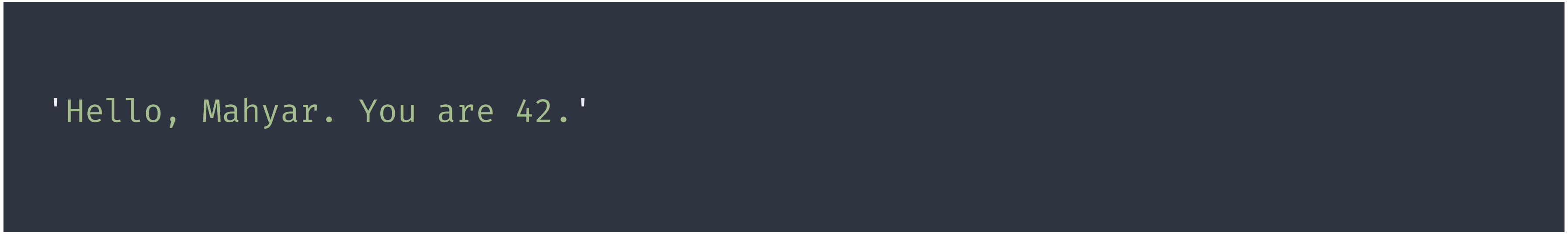
“Old-school” String Formatting: `str.format()`

INPUT:



```python
name = "Mahyar"
age = 42
print("Hello, {1}. You are {0}.".format(age, name))
```

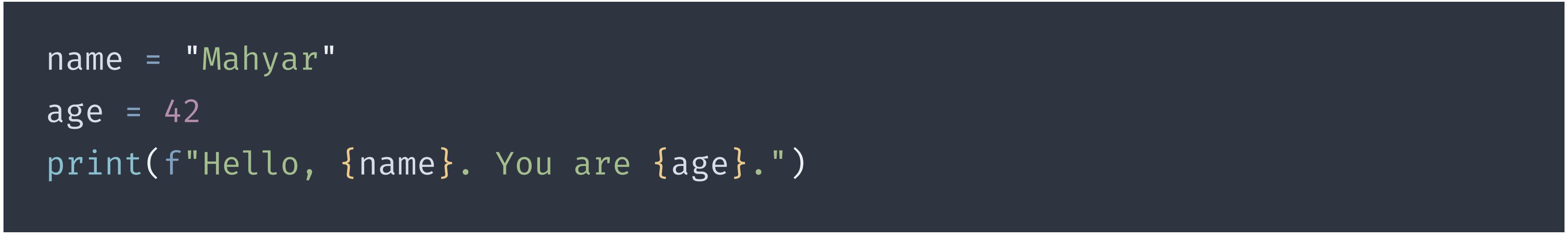
OUTPUT:



```
'Hello, Mahyar. You are 42.'
```

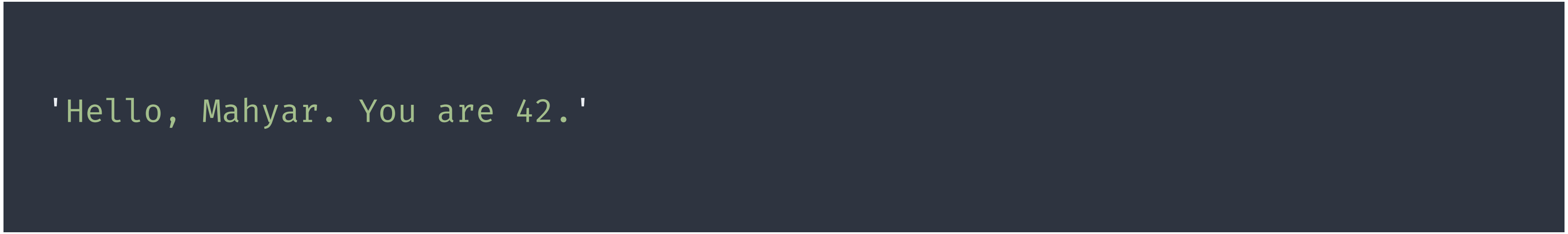
String Formatting using f-Strings

INPUT:



```python
name = "Mahyar"
age = 42
print(f"Hello, {name}. You are {age}.")
```

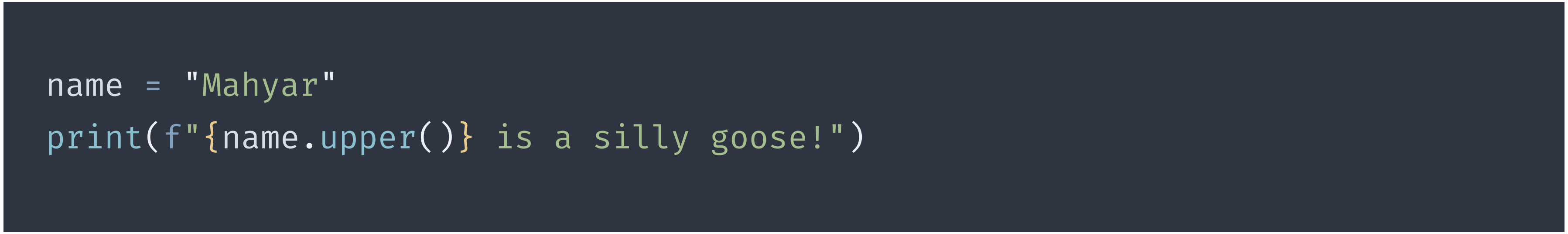
OUTPUT:



```
'Hello, Mahyar. You are 42.'
```

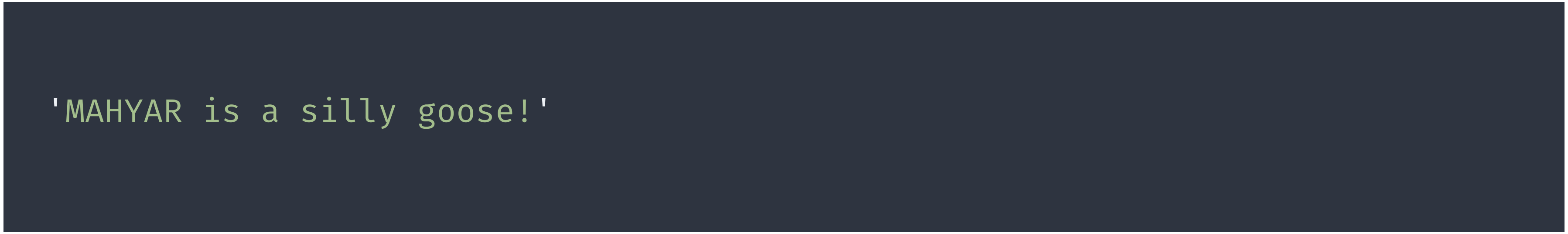
String Formatting using f-Strings

INPUT:



```python
name = "Mahyar"
print(f"{name.upper()} is a silly goose!")
```

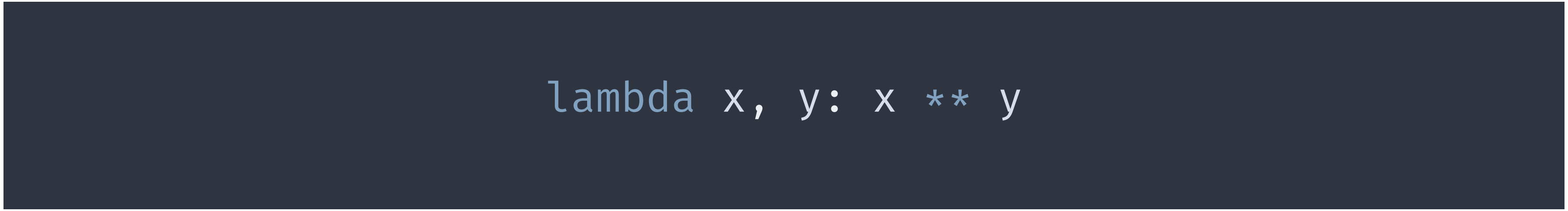
OUTPUT:



```
'MAHYAR is a silly goose!'
```

Lambda (Anonymous) Functions

Lambda Functions

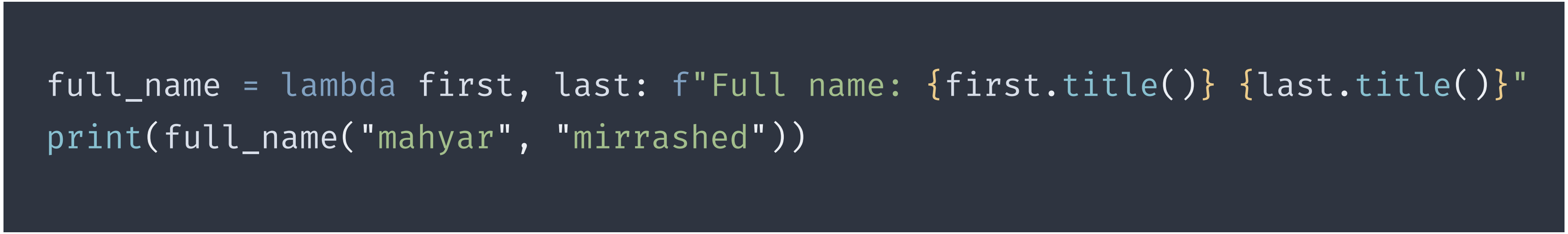


```
lambda x, y: x ** y
```

- Expression composition:
  - Keyword: `lambda`
  - Bound variable: `x, y`
  - Body: `x ** y`

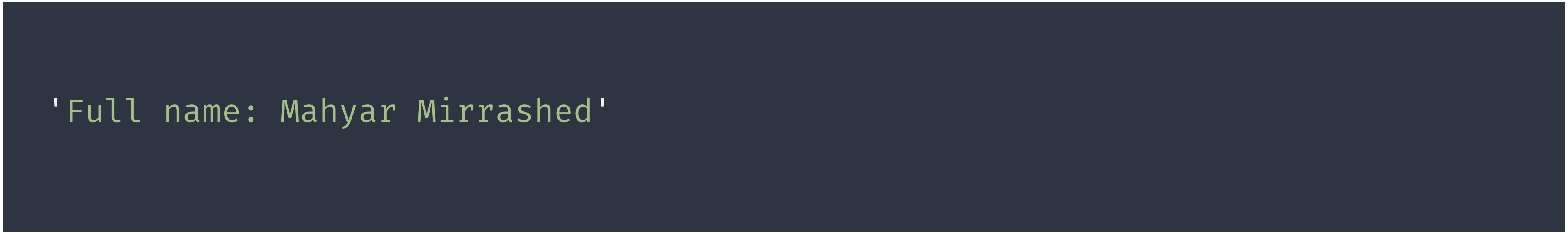
Named Lambda Functions

INPUT:



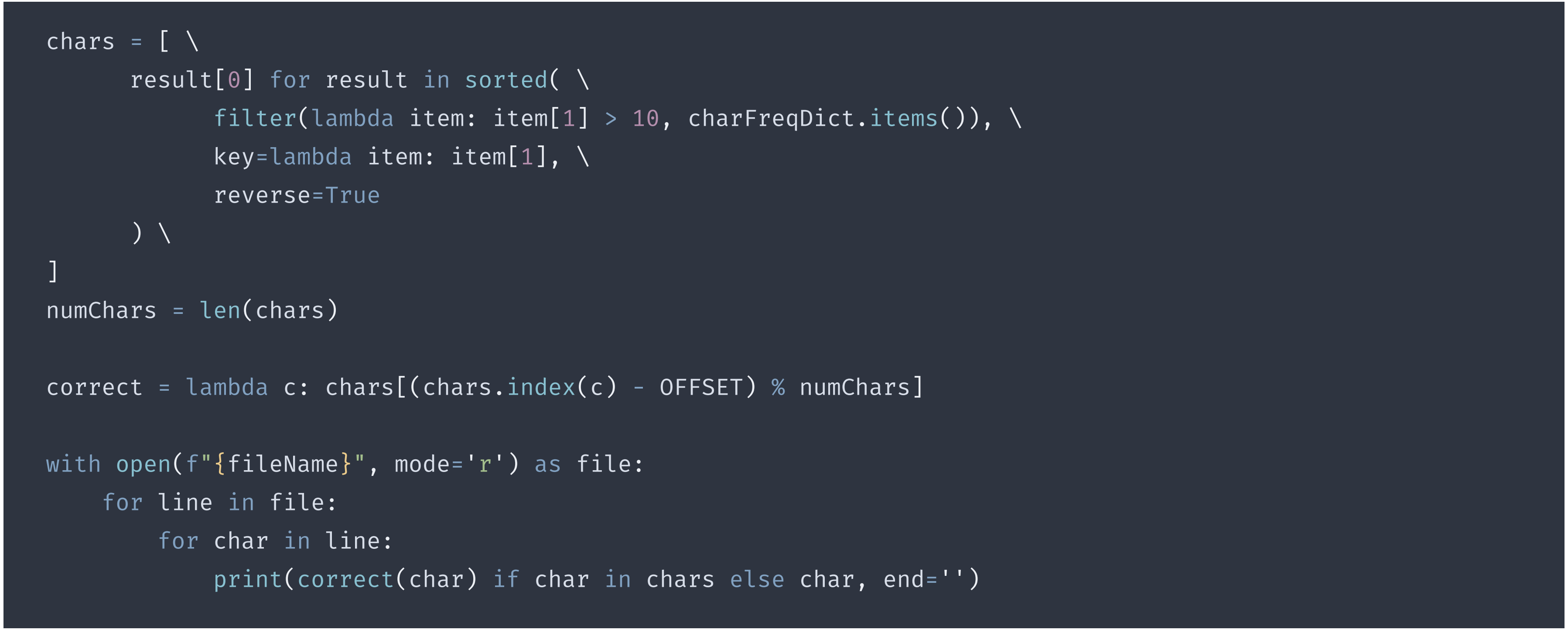
```python
full_name = lambda first, last: f"Full name: {first.title()} {last.title()}"
print(full_name("mahyar", "mirrashed"))
```

OUTPUT:



```
'Full name: Mahyar Mirrashed'
```

Example Lambda Usage in Assignments



```
chars = [ \
      result[0] for result in sorted( \
            filter(lambda item: item[1] > 10, charFreqDict.items()), \
            key=lambda item: item[1], \
            reverse=True
      ) \
]
numChars = len(chars)

correct = lambda c: chars[(chars.index(c) - OFFSET) % numChars]

with open(f"{fileName}", mode='r') as file:
    for line in file:
        for char in line:
            print(correct(char) if char in chars else char, end='')
```

Lambda Functions

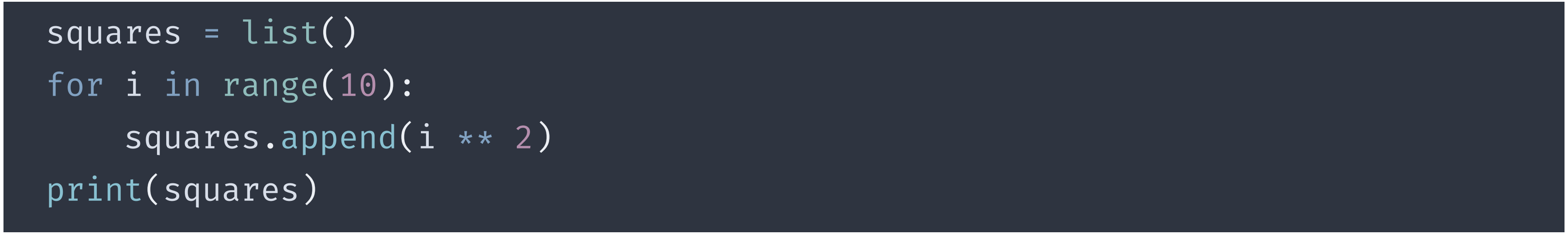
For interest only, no need to memorize:

- Anonymous functions
- Lambda functions
- Lambda expressions
- Lambda abstractions
- Lambda form
- Function literals

List, Dictionary, and Set Comprehensions

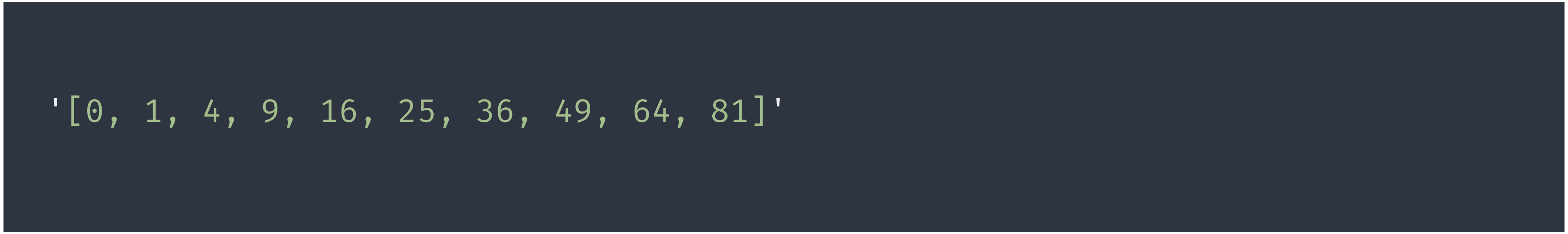
Standard Approach Using `for` Loop

INPUT:



```python
squares = list()
for i in range(10):
    squares.append(i ** 2)
print(squares)
```

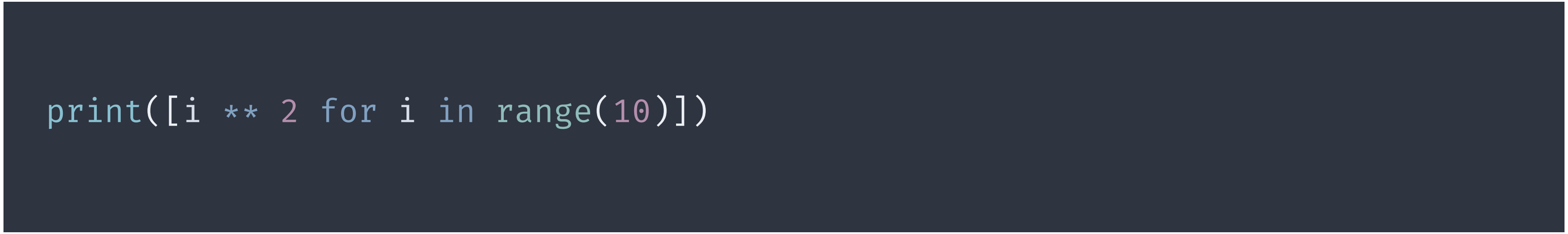
OUTPUT:



```
'[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]'
```

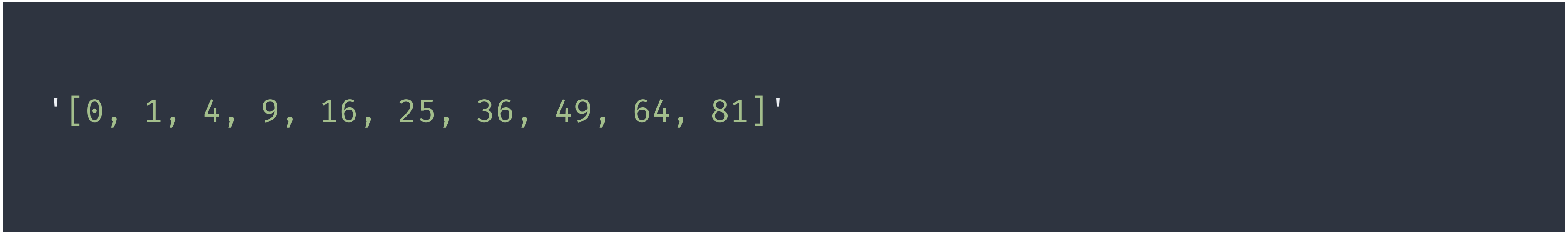
Innovative Approach Using List Comprehension

INPUT:



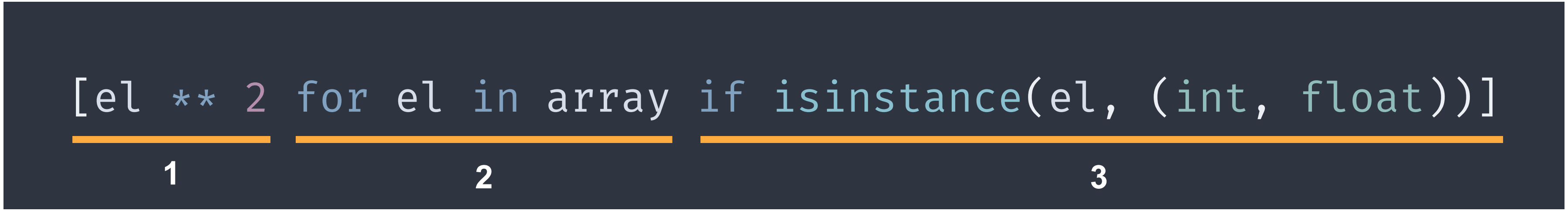
```
print([i ** 2 for i in range(10)])
```

OUTPUT:



```
'[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]'
```

List Comprehension Format



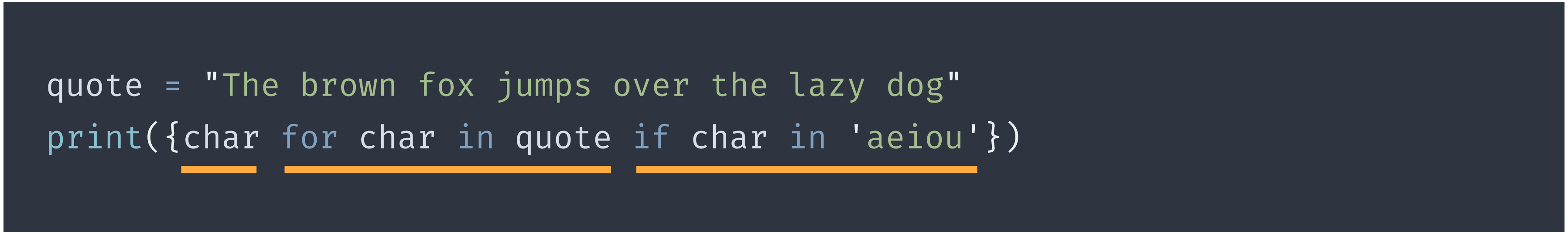
```python
[el ** 2 for el in array if isinstance(el, (int, float))]
```

List comprehension segments:

1) Expression
2) Iteration over iterable
3) Optional predicate

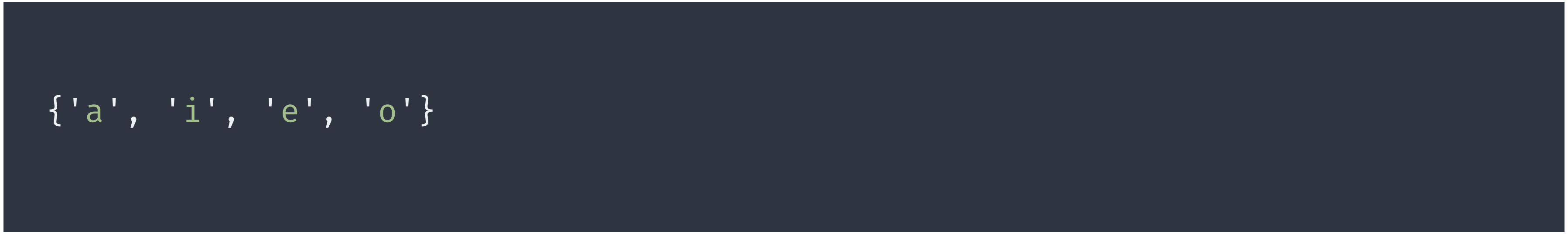
Set Comprehensions

INPUT:



```python
quote = "The brown fox jumps over the lazy dog"
print({char for char in quote if char in 'aeiou'})
```

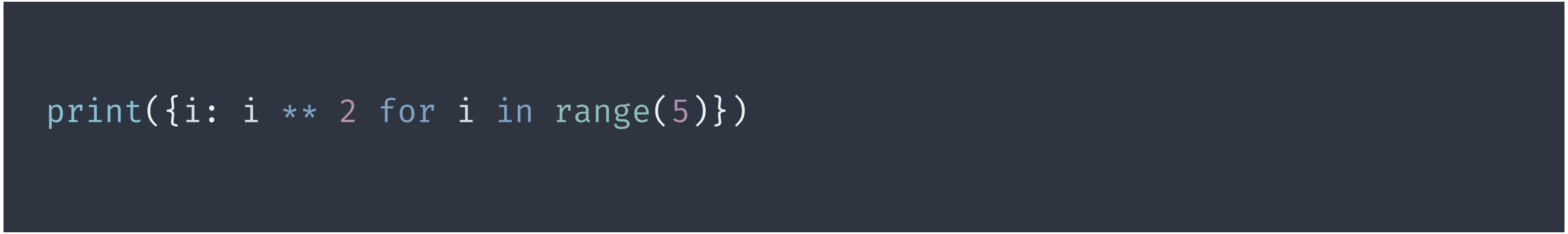
OUTPUT:



```
{'a', 'i', 'e', 'o'}
```

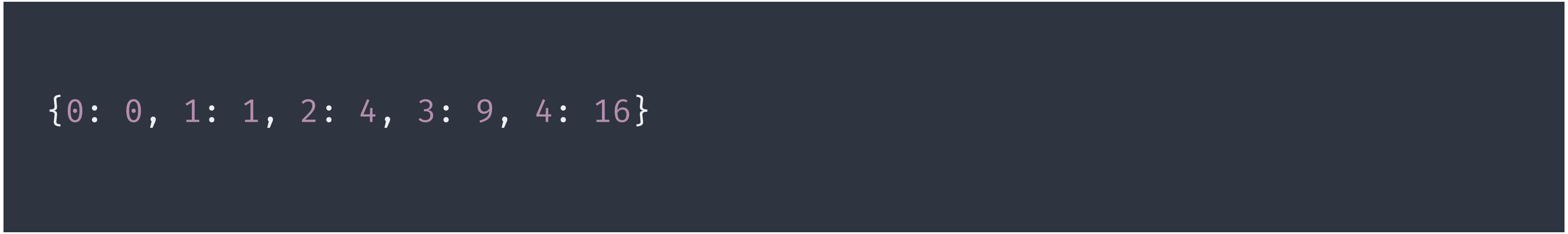
Dictionary Comprehensions

INPUT:



```python
print({i: i ** 2 for i in range(5)})
```

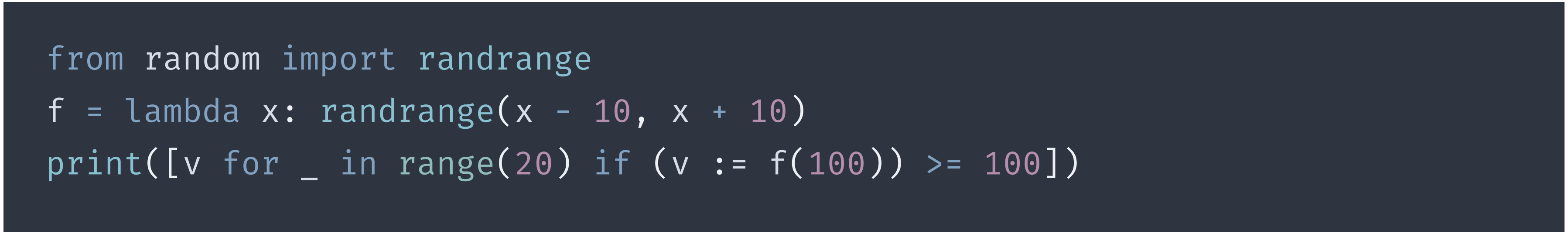
OUTPUT:



```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

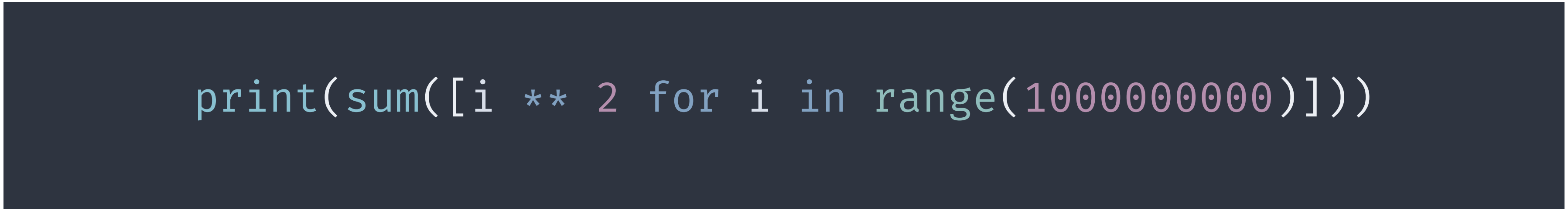
Interesting Applications of Comprehensions

- Comprehensions can be nested within each other
  - Useful for creating matrices
  - Useful for flattening matrices
- Ability to combine with Python 3.8 walrus operator
  - Allows to run expression while simultaneously assigning output value to a variable



```python
from random import randrange
f = lambda x: randrange(x - 10, x + 10)
print([v for _ in range(20) if (v := f(100)) >= 100])
```

Shortcomings of Comprehensions

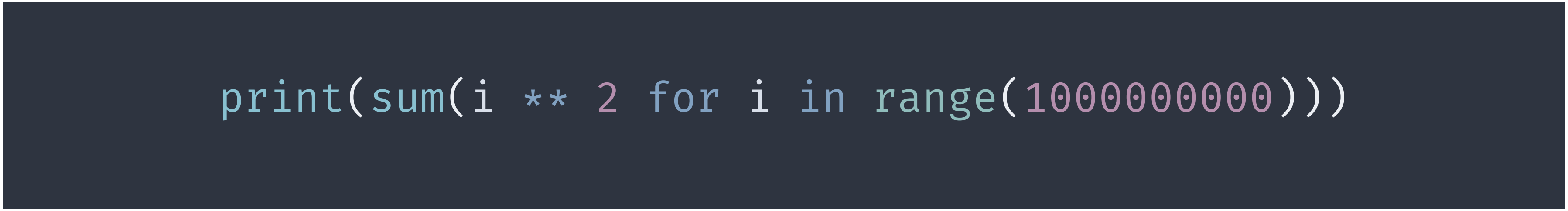


```
print(sum([i ** 2 for i in range(1000000000)]))
```

Starts by creating list of first billion perfect squares.

THEN, applies the summation over the entire list.

Generators

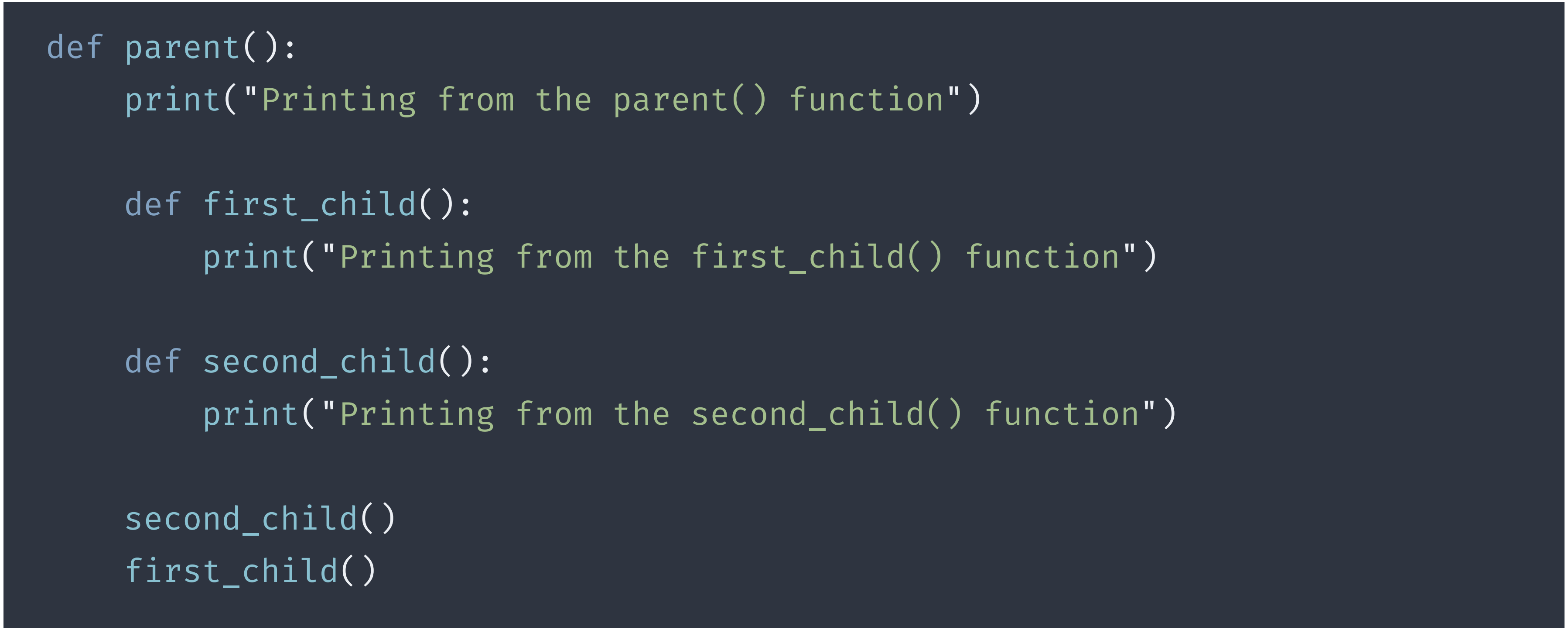


```python
print(sum(i ** 2 for i in range(1000000000)))
```

- Performs operations lazily (values only calculated when explicitly requested).
    - Keeps memory footprint small on machine

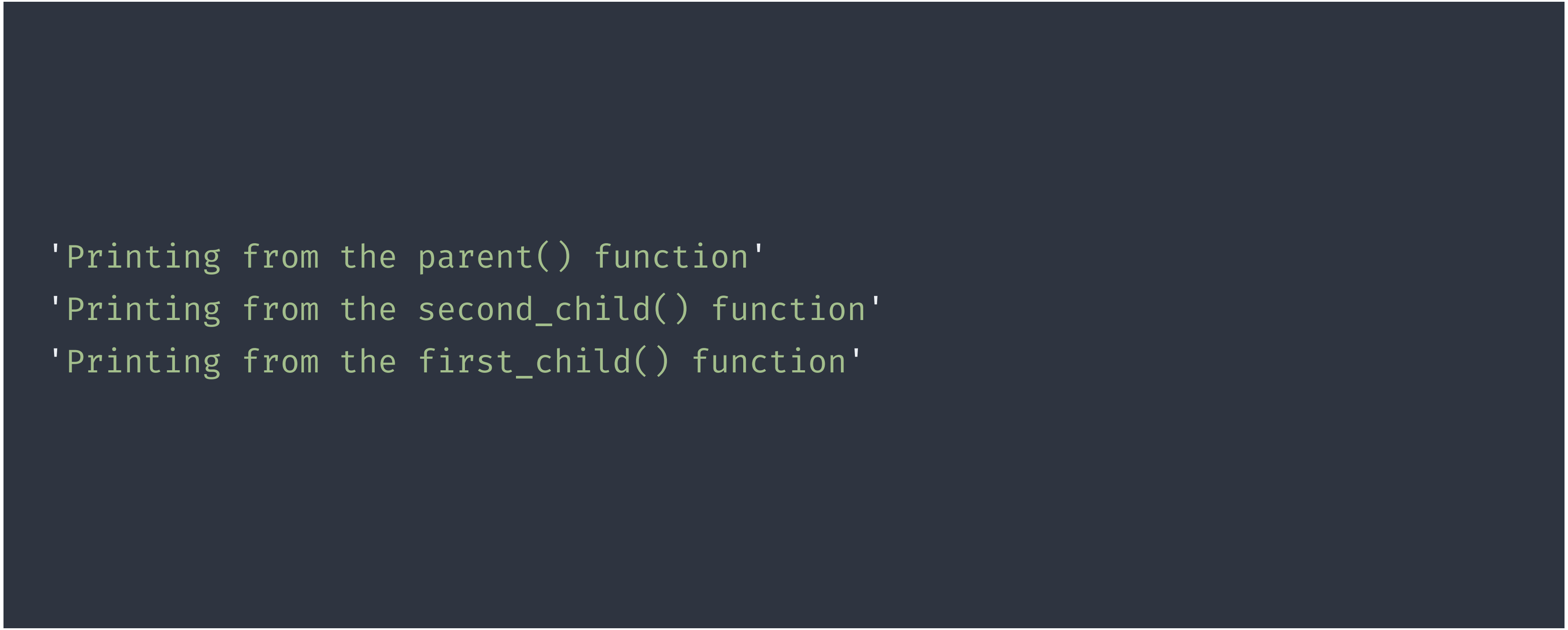
Function Decorators and Inner Functions

Inner Functions



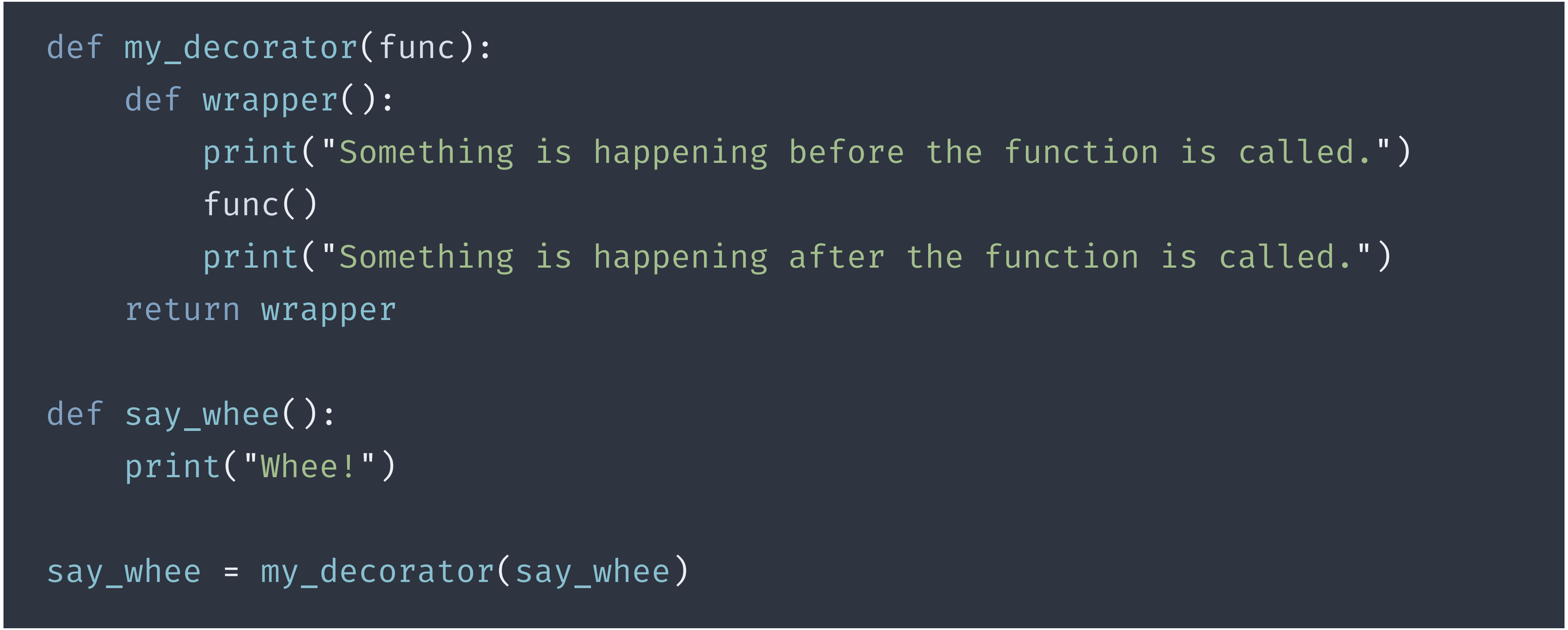
```python
def parent():
    print("Printing from the parent() function")

    def first_child():
        print("Printing from the first_child() function")

    def second_child():
        print("Printing from the second_child() function")

    second_child()
    first_child()
```

Inner Functions



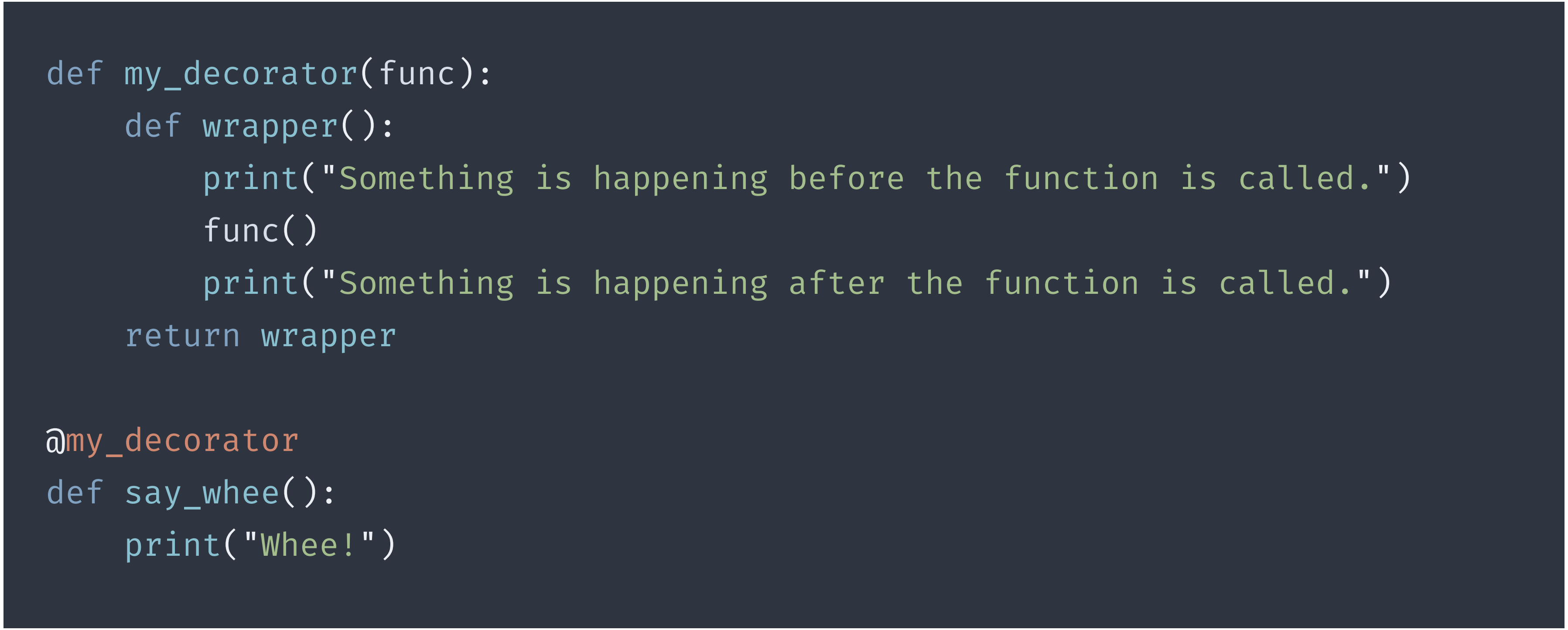
```
'Printing from the parent() function'
'Printing from the second_child() function'
'Printing from the first_child() function'
```

Function Decorators



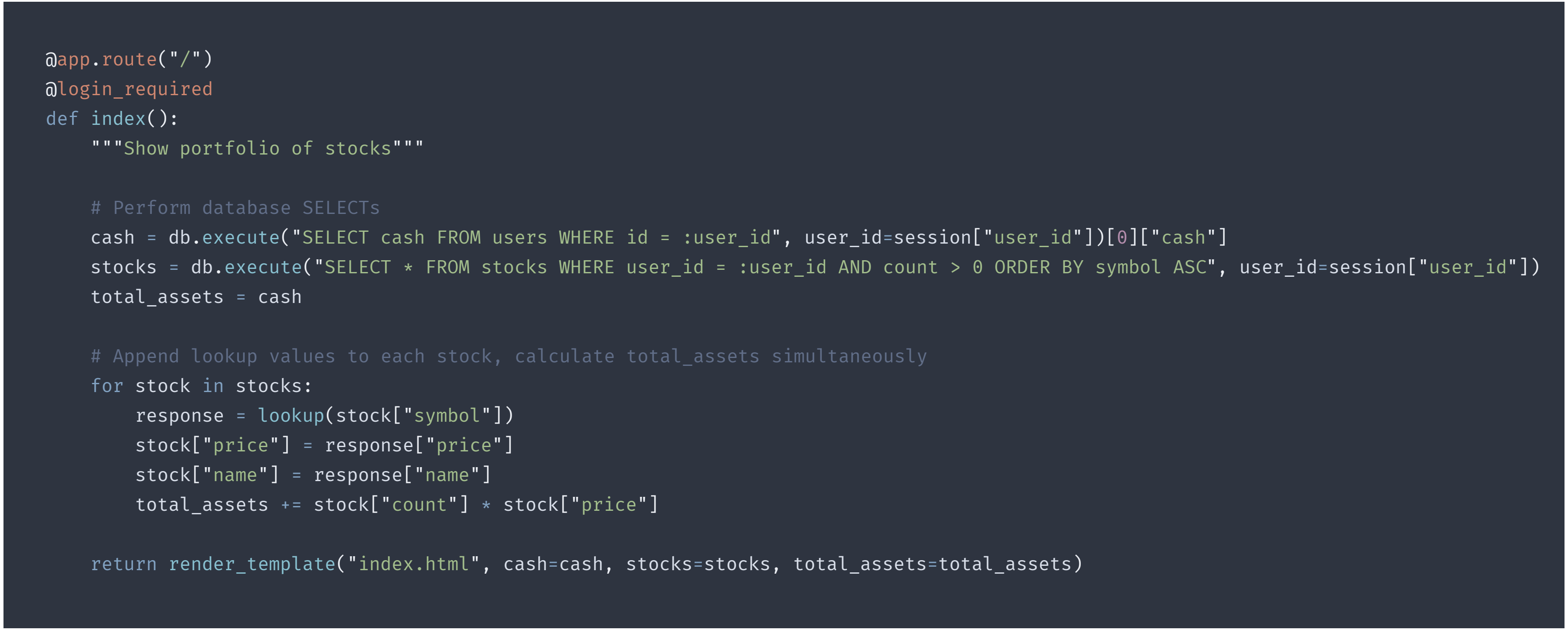
```python
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

def say_whee():
    print("Whee!")

say_whee = my_decorator(say_whee)
```

Function Decorators Using “Pie” Syntax



```python
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_whee():
    print("Whee!")
```

Function Decorators In Practice



```python
@app.route("/")
@login_required
def index():
    """Show portfolio of stocks"""

    # Perform database SELECTs
    cash = db.execute("SELECT cash FROM users WHERE id = :user_id", user_id=session["user_id"])[0]["cash"]
    stocks = db.execute("SELECT * FROM stocks WHERE user_id = :user_id AND count > 0 ORDER BY symbol ASC", user_id=session["user_id"])
    total_assets = cash

    # Append lookup values to each stock, calculate total_assets simultaneously
    for stock in stocks:
        response = lookup(stock["symbol"])
        stock["price"] = response["price"]
        stock["name"] = response["name"]
        total_assets += stock["count"] * stock["price"]

    return render_template("index.html", cash=cash, stocks=stocks, total_assets=total_assets)
```

Closing Remarks

Consider joining UMIEEE!

edu.ieee.org/ca-umieee/about-ieee/join-ieee-2/

UM IEEE

Questions?

Thanks for attending our workshop!

We will now take any additional remaining questions! (Please place in the chat.)

If you have more questions, join UMIEEE.